* Place dots on the diagram so that the Op codes will be properly decoded into instructions

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Instruction** | **Op5-Op0** | | addi | 001000 | | addiu | 001001 | | andi | 001100 | | ori | 001101 | |  |

Error in book page b-11 E=(A . B . !C)+(A . C . !B) . (B . C . !A)

^ should be +

* Plot the Roofline graph for memory bandwidth = 128.0 and Max FLOPS = 362.0 GFLOPS



* Provide the 3-hex-digit Hamming code for the byte 0x45

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

* Draw the instruction memory box and its input(s) and output(s).
* Indicate the mux signals supplied to the Forward A and Forward B muxes in this instruction execution sequence. For ease of grading. Just eliminate the ”00” mux signals.

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Forward A** | **Forward B** |
| Add $s7, $s1, $s4 |  |  |
| Add $s1, $s1, $s2 |  |  |
| Add $s0, $s7, $s3 |  |  |
| Add $s7, $s4, $s5 |  |  |
| Add $s5, $s2, $s0 |  |  |
| Add $s6, $s7, $s7 |  |  |
| Add $s2, $s4, $s2 |  |  |
| Add $s6, $s5, $s7 |  |  |
| Add $s6, $s6, $s5 |  |  |
| Add $s5, $s4, $s3 |  |  |

* Show the execution steps of the following instructions:
* Without forwarding

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| addi $s1 $s0 1 | If | id | ex | mem | wb |  |  |  |  |  |  |  |  |  |
| add $s2 $s2 $s2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| add $s3 $s2 $s3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| add $s4 $s1 $s2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sw $s2 0($s3) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

* With forwarding

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| addi $s1 $s0 1 | If | id | ex | mem | wb |  |  |  |  |  |  |  |  |  |
| add $s2 $s2 $s2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| add $s3 $s2 $s3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| add $s4 $s1 $s2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sw $s2 0($s3) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

* Fill in the blanks

|  |  |  |
| --- | --- | --- |
| **Hexadecimal** | **Decimal** | **Binary** |
|  |  | 11.01 |
|  |  | 1110.111 |
| 0xe.8 |  |  |
| 0x0.a |  |  |
|  | 15.750 |  |

* Write the following instruction in Hex: SUB $t6, $t0, $s5

* Calculate the effective CPI;

|  |  |  |  |
| --- | --- | --- | --- |
| **Inst Class** | **A** | **B** | **C** |
| **CPI** | 1 | 4 | 2 |
| **Proportion** | 1/6 | 1/3 | 1/2 |

* Order the machines from fastest to slowest:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Machine** | **Instructions** | **CPI** | **Clock (GHz** | **Exec Time** |
| **A** | 1900 | 1.4 | 1.4 |  |
| **B** | 1300 | 1.5 | 1.5 |  |
| **C** | 1400 | 1 | 2 |  |
| **D** | 2000 | 2.1 | 3.5 |  |

\_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_

(slow) (fast)

**ADD -- *Add***

|  |  |
| --- | --- |
| Description: | Adds two registers and stores the result in a register |
| Operation: | $d = $s + $t; advance\_pc (4); |
| Syntax: | add $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0000 |

**ADDI -- *Add immediate (with overflow)***

|  |  |
| --- | --- |
| Description: | Adds a register and a sign-extended immediate value and stores the result in a register |
| Operation: | $t = $s + imm; advance\_pc (4); |
| Syntax: | addi $t, $s, imm |
| Encoding: | 0010 00ss ssst tttt iiii iiii iiii iiii |

**ADDIU -- *Add immediate unsigned (no overflow)***

|  |  |
| --- | --- |
| Description: | Adds a register and a sign-extended immediate value and stores the result in a register |
| Operation: | $t = $s + imm; advance\_pc (4); |
| Syntax: | addiu $t, $s, imm |
| Encoding: | 0010 01ss ssst tttt iiii iiii iiii iiii |

**ADDU -- *Add unsigned (no overflow)***

|  |  |
| --- | --- |
| Description: | Adds two registers and stores the result in a register |
| Operation: | $d = $s + $t; advance\_pc (4); |
| Syntax: | addu $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0001 |

**AND -- *Bitwise and***

|  |  |
| --- | --- |
| Description: | Bitwise ands two registers and stores the result in a register |
| Operation: | $d = $s & $t; advance\_pc (4); |
| Syntax: | and $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0100 |

**ANDI -- *Bitwise and immediate***

|  |  |
| --- | --- |
| Description: | Bitwise ands a register and an immediate value and stores the result in a register |
| Operation: | $t = $s & imm; advance\_pc (4); |
| Syntax: | andi $t, $s, imm |
| Encoding: | 0011 00ss ssst tttt iiii iiii iiii iiii |

**OR -- *Bitwise or***

|  |  |
| --- | --- |
| Description: | Bitwise logical ors two registers and stores the result in a register |
| Operation: | $d = $s | $t; advance\_pc (4); |
| Syntax: | or $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0101 |

**ORI -- *Bitwise or immediate***

|  |  |
| --- | --- |
| Description: | Bitwise ors a register and an immediate value and stores the result in a register |
| Operation: | $t = $s | imm; advance\_pc (4); |
| Syntax: | ori $t, $s, imm |
| Encoding: | 0011 01ss ssst tttt iiii iiii iiii iiii |

**SB -- *Store byte***

|  |  |
| --- | --- |
| Description: | The least significant byte of $t is stored at the specified address. |
| Operation: | MEM[$s + offset] = (0xff & $t); advance\_pc (4); |
| Syntax: | sb $t, offset($s) |
| Encoding: | 1010 00ss ssst tttt iiii iiii iiii iiii |

**SLL -- *Shift left logical***

|  |  |
| --- | --- |
| Description: | Shifts a register value left by the shift amount listed in the instruction and places the result in a third register. Zeroes are shifted in. |
| Operation: | $d = $t << h; advance\_pc (4); |
| Syntax: | sll $d, $t, h |
| Encoding: | 0000 00ss ssst tttt dddd dhhh hh00 0000 |

**SLT -- *Set on less than (signed)***

|  |  |
| --- | --- |
| Description: | If $s is less than $t, $d is set to one. It gets zero otherwise. |
| Operation: | if $s < $t $d = 1; advance\_pc (4); else $d = 0; advance\_pc (4); |
| Syntax: | slt $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 1010 |

**SLTI -- *Set on less than immediate (signed)***

|  |  |
| --- | --- |
| Description: | If $s is less than immediate, $t is set to one. It gets zero otherwise. |
| Operation: | if $s < imm $t = 1; advance\_pc (4); else $t = 0; advance\_pc (4); |
| Syntax: | slti $t, $s, imm |
| Encoding: | 0010 10ss ssst tttt iiii iiii iiii iiii |

**SLTU -- *Set on less than unsigned***

|  |  |
| --- | --- |
| Description: | If $s is less than $t, $d is set to one. It gets zero otherwise. |
| Operation: | if $s < $t $d = 1; advance\_pc (4); else $d = 0; advance\_pc (4); |
| Syntax: | sltu $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 1011 |

**SRL -- *Shift right logical***

|  |  |
| --- | --- |
| Description: | Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in. |
| Operation: | $d = $t >> h; advance\_pc (4); |
| Syntax: | srl $d, $t, h |
| Encoding: | 0000 00-- ---t tttt dddd dhhh hh00 0010 |

**SUB -- *Subtract***

|  |  |
| --- | --- |
| Description: | Subtracts two registers and stores the result in a register |
| Operation: | $d = $s - $t; advance\_pc (4); |
| Syntax: | sub $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0010 |

**SW -- *Store word***

|  |  |
| --- | --- |
| Description: | The contents of $t is stored at the specified address. |
| Operation: | MEM[$s + offset] = $t; advance\_pc (4); |
| Syntax: | sw $t, offset($s) |
| Encoding: | 1010 11ss ssst tttt iiii iiii iiii iiii |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Register Number** | **Usage** | **Preserved by callee?** |
| $zero | 0 | hardwired 0 | N/A |
| $v0-$v1 | 2-3 | return value and expression evaluation | no |
| $a0-$a3 | 4-7 | arguments | no |
| $t0-$t7 | 8-15 | temporary values | no |
| $s0-$s7 | 16-23 | saved values | YES |
| $t8-$t9 | 24-25 | more temporary values | no |
| $gp | 28 | global pointer | YES |
| $sp | 29 | stack pointer | YES |
| $fp | 30 | frame pointer | YES |
| $ra | 31 | return address | YES |